Introduction to Neural Networks and Deep Learning

Syllabus

1. Introduction to Deep Learning
   1. Introduction to Deep Learning
   2. Biological Neural Networks and Deep Learning
   3. Artifiicial Neural Networks – Forward Propagation
2. Artificial Neural Networks
   1. Gradient Descent
   2. Backpropagation
   3. Vanishing Gradient
   4. Activation Functions
3. Deep Learning Librabries
   1. Introduction to deep learning libraries
   2. Regression models with Keras
   3. Classification Models with Kera
4. Deep Learning Models
   1. Shallow and Deep Neural Networks+
   2. Convolutional Neural Networks
   3. Recurrent Neural Networks
   4. Autoencoders

**Deep Learning with Keras**

Applications with DL

* Color restoration
* Speech Reenactment
* Handwriting
* Automatic Machine translation
* Object classification

Neurons and Neural Networks

Many algorithms in DL are inspired by the way neurons and neural network’s function.

Artificial Neurons behave in the same way as biological neurons

Artificial Neurall Networks

Consist of layers

* Input layer
* Output layer
* Hidden layers

Main topics

* Forward propagation
* Backward propagation
* Activation Functions

Forward propagation

The process of data pass through layer of neurons in a neural network from input to output

Normally represented as x1 \* w1 + xn \* wn + b1 = a = z

Where x represents the input value and w represents the weighting or coefficient and b represents the bias (a constant), z is the combination of weioghts, coefficients and bias, and a is the output of the network. NB. When using multiple layers, the output from the previous layers neuron is the input for the current layer.

Simply outputting a weighted sum of inputs is limiting. So the sum is normally mapped to a nonlinear space often using the sigmoid function.

Nonlinear functions like sigma are called **activation functions.** They (more or less) decide whether a neuron should be active or not based on the relevance of the information being received.

A neural network without an activation function is ostensibly just a linear regression model.

**Gradient Descent**

Cost Function

To decide in which direction to move (which side of the curve are we on? Is pos or neg?) we use the slope of the tangent at or:

The magnitude of the movement along is determined by the learning rate.

Higher the learning rate the bigger the step in the cost function

So

Process is repeated using same learning rate until minimum is hit (or as close as)

Nb. Large and small steps have positives and negatives. Ie speed vs. accuracy.

The goal of the algorithm in the neural network is to minimize the loss function by updating the weights of the network iteratively. The objective then is to find the set of weights that results in the lowest possible value of the loss function. What this is indicates is how accurate the model predicts the target values.

Several algorithms are commonly used variations of the basic gradient descent algorithm to enhance its performance and address some of its limitations. Some of the most widely used gradient descent algorithms include:

1. **Batch Gradient Descent**: This is the standard form of gradient descent. In each iteration, the algorithm computes the gradient using the entire training dataset. It can be slow for large datasets but can converge to a global minimum if the loss function is convex.
2. **Stochastic Gradient Descent (SGD)**: In each iteration, SGD randomly selects a single training example and computes the gradient based on that example. It can be much faster than batch gradient descent and can escape local minima more easily. However, its updates can be noisy, and the convergence can be erratic.
3. **Mini-Batch Gradient Descent**: This is a compromise between batch gradient descent and SGD. In each iteration, a small random subset (mini-batch) of the training data is used to compute the gradient. This combines some of the benefits of both batch and stochastic approaches.
4. **Gradient Descent with Momentum**: This algorithm uses an exponentially decaying moving average of past gradients to accelerate convergence, especially in directions with consistent gradients. It helps overcome oscillations and speeds up convergence.
5. **Adaptive Learning Rate Methods**:
   * **Adagrad**: Adapts the learning rate individually for each parameter by dividing the learning rate by the square root of the sum of squared gradients.
   * **RMSProp**: Similar to Adagrad but with a decaying average of squared gradients, preventing the learning rate from getting too small.
   * **Adam (Adaptive Moment Estimation)**: Combines momentum and adaptive learning rates. It maintains moving averages of both gradients and squared gradients and adjusts the learning rate accordingly.
6. **Nesterov Accelerated Gradient (NAG)**: This is an improvement upon the standard momentum approach. It adjusts the momentum term to take into account a "lookahead" gradient computed at a slightly ahead position, resulting in faster convergence.
7. **Rprop (Resilient Propagation)**: Updates the learning rates based on the sign of the gradient. If the gradient keeps the same sign, the learning rate increases; if it changes sign, the learning rate decreases.
8. **Adaptive Subgradient Methods**:
   * **AdaGrad**: Adapts the learning rate for each parameter based on the magnitude of historical gradients.
   * **Adadelta**: An extension of AdaGrad that addresses the issue of diminishing learning rates by using a running average of squared parameter updates.

These are just a few of the many variations and improvements to the basic gradient descent algorithm. The choice of algorithm depends on factors like the problem, the size of the dataset, the network architecture, and desired convergence properties.

**Back Propagation**

How do models optimize weights and biases?

Model training is done in a supervised setting where each data point has a label or ‘ground truth’ (T)

Training is needed when the predicted value != ground truth

The training process follows these steps.

1. Calculate the error (E) between the ground truth and the output. This error represents the cost/loss function.
2. The error is then propagated back into the network and each weight and bias is updated per the gradient descent equations, ie:

The error is calculated using mean squared error, then propagated back into the network.

For example, in a network with 2 nodes and paths (b1, w1, b2, w2) to update w2 the gradient descent equation is used. However E is not explicitly a function of w2, the chain rule must be used to establish the derivative of the error with respect to w2.

E is a function of a2

A2 is a function of z2

And z2 is a function of w2

Therefore, we can take the derivative of E with respect to a2

The derivative of a2 with respect to z2

And the derivative of z2 with respect to w2

Then the derivative of the error with respect to w2 would be the product of these individual derivatives.

**What is chain rule?**

The chain rule states that the derivative of a composite function (a function formed by applying a function to the output of another function) is the product of the derivatives of the component functions ie:

If we have 2 function f and g that combine to function h:

then (using Leibniz notation)

To summarize

1. Init weights and biases
2. Calculate network output using forward propacgation
3. Calculate error between ground truth and estimated or predicted output
4. Update weights and biases through back propagation
5. Repeat steps 2 – 4 until a number of iterations/epochs are reached or error between ground truth and predicted out is below a predefined threshold.

**Vanishing Gradient**

A problem with the sigmoid activation function.

The size of the error with respect to w1 is incredibly small.

This is because factors of less than one (due to the sigmoid activation) are multiplied repeadetdly resulting in smaller and smaller.

This means neurons at the earlier layers decrease to the lowest error more slowly than the later layers and some too slowly to make the model usable.

So the sigmoid function is not used for activation.

**Activation Functions**

The functions that are now used

Types:

1. Binary Step Function
2. Linear Function
3. Sigmoid Function
4. Hyperbolic Tangent Function
5. ReLu (Rectified Lienear Unit)
6. Leaky ReLu
7. Softmax Function

Of these, 4 are most commonly used:

* + The sigmoid
  + Hyperbolic Tangent Function
  + ReLu
  + Softmax

**Sigmoid Function**

At z = 0 x = 0.5

When z is large positive number a is close to 1 and vice versa

The function is however normally flat beyond +3 and -3

This means that once the function enter these regions, the gradients become very small

Resulting in the vanishing

gradient problem.

Similar the values received are always positive and not symmetric around the origin.

By scaling the sigmoid function, this problem can be avoided, resulting in the hyperbolic tangent function.

**Hyperbolic Tangent Function**

Simialar to the sigmoid function, it is however symmetrical round the origin and the values range from -1 to 1.

It also however leads to the vanishing gradient problem in very deep neural networks

**Rectified Linear Unit ReLu**

The most widely used activation function

Its nonlinear, it doesn’t activate all neurons at the same time.

If the input is negitave the output is converted to 0 and the neuron is not activated.

This means that only a few neurons are active at any one time making the network very efficient.

It does not suffer from the vanishing gradient.

**Softmax Function**

Useful for classification problems.

Ideally used in the output layer of the classifier to find the probability of each output.

Eg for a vector of outputs.

In summary, the sigmoid and hyperbolics tangent function are avoided in most applications due to the vanishing gradient problem. Instead ReLu is the function that is most often used BUT only in the hidden layers.

Popular Deeplearning Librabries and Frameworks

1. Tensorflow (google)
2. Keras (also google)
3. PyTorch (Version of the Lua-based torch framework) supports machine learning algorithms running on GPUs in particular.
4. Theano (No longer supported)

Both pytorch and tensorflow have a steep learning curve. Keras has apparently the easiest API and go-to library for quick prototyping and fast development times.

It’s a high-level API.

Keras runs on top of a high level library like tensorflow.

**Regression Models with Keras**